第 3 章　轻量级流密码

3.1　概述

3.1.1　流密码一般性设计原理

1949 年，Shannon 证明了一次一密体制在唯密文攻击下是理论上不可破译和绝对安全的；然而，为了建立一次一密的密码系统，通常需要在安全信道上交换传输至少和明文一样长的密钥，在很多情况下是不现实和不经济的，并且在密钥的产生和管理方面也面临着许多复杂问题。流密码算法本质上是对一次一密体制的模仿，同时消除了密钥产生、分配和管理维护中的各类问题。流密码所产生的密钥流至少要做到看起来很像随机比特序列，且要求恢复算法的初始状态和密钥或者将算法及其密钥流与随机情况区分开来都是在一定计算能力与许可范围内困难的。

流密码一般可分为同步流密码和自同步流密码两大类。在同步流密码中，生成的密钥流和发送的明文消息之间相互独立，其内部状态仅仅依赖于上一时刻的内部状态，与输入明文无关。同步流密码的优点在于其有限的错误传播，当一个符号在传输过程中发生错误后不会影响后续的符号。而自同步流密码的密钥流则依赖之前的明/密文信息，常见的自同步流密码是类似于分组密码密文反馈模式的自同步流密码，即密文参与密钥流的生成过程，这使得这类流密码的安全性从理论上分析起来非常困难，也造成这类算法的设计非常稀少。目前常见的大多数流密码算法都是同步流密码，因其设计上的可分析性，使得设计者能够更好地理解自己设计的流密码对于已知甚至某些未知攻击的抵抗力。从总体上说，一个流密码的安全性在很大程度上取决于其采用的密钥流生成器，由于以线性反馈移位寄存器（LFSR）为研究对象的伪随机序列代数理论的成熟，20 世纪五六十年代以来的大量流密码多是基于 LFSR 设计，例如美国未公开的 Fibonacci 生成器、E0、A5/1 等。由于 LFSR 的线性性质对于密码分析没有任何免疫力（如 Berlekamp-Massey 算法等），需要采用各种非线性部件来显式或隐式地掩盖线性性质并增强其非线性，常见的方法有非线性组合、非线性滤波、不规则钟控、带记忆及其各种组合方式。Rueppel 给出了此类流密码设计的一些方法和准则，比如著名的线性驱动部件加非线性组合部件的设计范式[285]。这些方法和准则代表了传统的流密码设计思路及其衍生准则，其中一些已经很少见，另一些的影响则仍然存在。21 世纪伊始，由于代数攻击的提出，传统的基于 LFSR 的流密码设计方案进一步丧失了其吸引力。

2000 年，欧洲提出了 NESSIE（New European Schemes for Signatures, Integrity, and Encryption，欧洲新签名、完整性和加密方案）计划。这个计划一直持续到 2003 年，其目标之一就是征集新的流密码算法，但很遗憾的是，研究者提交的所有流密码算法，包括较为著名的 SNOW1.0、LILI-128 等，在安全性方面都存在一些问题，最终没有一个流密码能够入选。为了进一步推动流密码的发展，2004 年欧洲启动了 ECRYPT 计划，其中就包含了专门面向流密码的 eSTREAM 计划，其主要目标是征集一些新的流密码算法，以便于日后广泛

应用。eSTREAM 计划共征集到 34 个算法，总体来看，算法设计思路多样化，且大多数不再局限于传统的基于 LFSR 的结构，而是采用了一些新的设计思路，比如采用非线性反馈移位寄存器、采用类似分组密码的结构等。具体来讲，eSTREAM 计划将流密码算法分为两大类：面向受限硬件环境的算法，比如 Grain v1、Trivium 和 Mickey v2 等，这些算法一般采用比特级的操作；面向高速软件应用的算法，比如 Salsa20/12、SOSEMANUK、Rabbit 和 HC 等[286,287]，这些算法大都采用面向字的操作。eSTREAM 计划明显地影响了流密码的发展进程，其后流密码的研究很快陷入沉寂，这主要是由于学术界对于大量采用极大内部状态和极高初始化轮数的新型流密码算法普遍缺乏有效的分析思路和办法，这一状况一直持续到现在。其间虽有立方攻击、扩域上快速相关攻击等新方法的提出，但对于采用极大内部状态和极高初始化轮数的流密码，短时间内还是无法获得快于穷搜索的攻击。

在 ESC 2013 上，一项新的对称密码算法竞赛项目被提出，称为 CAESAR 竞赛，其目的是征集认证加密算法。传统的单一目的对称密码算法及其组合在某些情况下已经无法满足实际应用的需求，新的应用趋势是在确保机密性的同时也要保证信息的完整性，这就是 CAESAR 竞赛的初衷。从提交的算法来看，基于流密码的候选算法的主要思想仍然是 eSTREAM 计划最终入选算法的组合与改进，变化只是在传统的加密模块之外增加的认证模块以及采用杂凑函数结构设计的流加密算法。最近，为了满足同态加密应用环境的需求，在 Eurocrypt 2016 上，一个新的流密码结构被提出，称为置换滤波生成器，这种结构可以看作滤波生成器的一个变体，它以人为的形式模拟产生大量固定次数、固定形式的代数等式，以满足同态加密应用环境的需求。

3.1.2 轻量级流密码研究进展

回顾历史可以发现，欧洲 eSTREAM 计划最终入选算法即包括了适用于硬件实现的轻量级流密码算法，比如 Grain v1、Trivium 及 Mickey v2 等。以 Grain v1 为例，它由 Martin Hell 等人设计，算法分为密钥流产生和初始化两个阶段，密钥长度为 80 比特，也有采用 128 比特密钥的 Grain-128 和 Grain-128a，这些流密码算法均比较适合 RFID 等资源受限环境。Trivium 算法虽然结构简单，但目前并没有发现明显的安全性漏洞，而且具有轻量级的显著特点。此外，还有一些专门为 RFID 标签设计的轻量级流密码，比如 WG-7 算法和 A2U2 算法。WG-7 算法的密钥长度为 80 比特，初始向量为 81 比特，是专门为 RFID 应用而设计的流密码算法。虽然设计者给出了一定前提下的安全性分析证明[288]，力图说明 WG-7 算法能够有效抵御差分分析、代数攻击、相关攻击等分析方法，但已经有一些密码分析结果表明该算法并不能提供 80 比特的安全性。A2U2 也是专用的轻量级流密码，它结合了流密码的设计原则和某些分组密码的设计方法，考虑了资源受限设备的特点，其硬件实现代价较低。但是，已有密码分析结果表明 A2U2 的设计并不成功。WG-7 和 A2U2 的例子可以说明，轻量级流密码的设计绝非易事。

纵观近年推出的轻量级流密码算法，可以观察到两条明显的技术路线：第一条路线采用较大的内部状态与相对简单的逻辑运算；第二条路线采用较小的内部状态与相对复杂的逻辑组合函数。第一条路线采用通常的流密码设计思路，尽可能降低算法的逻辑运算占用的硬件资源；第二条路线却反其道而行之，采用较小的内部状态，即内部状态小于密钥长度的两倍，但在密钥流生成阶段却采用了依赖于初始密钥的状态更新函数，这种轻量级流密码

现在一般称为小状态流密码。

第一条路线的一个典型代表是 CAESAR 竞赛入选算法 ACORN。虽然可以看出 ACORN 算法与 Grain 系列算法的明显关系，即 Grain 系列算法只是把非线性反馈加在中间，而 ACORN 算法则将反馈加在了最远的反馈端，但 ACORN 算法采用的逻辑运算比较简单，非线性运算只包含了数个逻辑与门，线性运算则有多个逻辑异或门，其扩散性由 6 个级联的线性移位寄存器来保证。在某些特定硬件平台上，该算法的实现较为简便。ACORN 允许多拍并行运行，即同时输出多个连续的密钥流比特，这一特点既适合硬件实现，也适合软件实现，可以说是同时面向软硬件的轻量级流密码算法。

第二条路线的轻量级流密码目前有多个典型算法，比如 Sprout、Plantlet 等。其核心设计思想是：在密钥流生成阶段，采用依赖于密钥的状态更新函数重复利用密钥，以打破时间/存储/数据折中曲线规定的复杂度限制，从而使得算法的内部状态可以小于密钥长度的两倍。在这一大类算法之下继续细分，还可以观察到一种变形设计——FP(1)模式[289]，它在初始化阶段采用了再次异或密钥的方式，使得从初始状态求逆以恢复密钥变得代价高昂，这种轻量级流密码算法在密钥流生成阶段并没有采用依赖于密钥的设计方案，而是像经典的流密码算法一样运行。从目前公开文献提出的算法来看，无论是哪种设计思路，其本质都是基于 Grain 系列算法的结构设计的。

在上述第二条路线的两种设计思路中，前一种设计思路的代表性轻量级流密码算法有 Sprout、Fruit 和 Plantlet 等。Sprout 算法于 2015 年首次提出采用依赖于密钥的轻量级流密码设计思想，试图在 Grain-128a 的基础上做一些避免后者弱点的修改，以更小的内部状态提供同等的时间/存储/数据折衷攻击免疫力，因此可以有效降低算法的硬件实现面积。Sprout 算法在结构上与 Grain 算法相似，它包括一个 40 比特的 NFSR、一个 40 比特的 LFSR、一个 80 比特的固定密钥寄存器和一个计数器。由于存储固定密钥要比一般的寄存器节省硬件面积，所以 Sprout 算法可以获取一定的硬件实现上的收益。但是 Sprout 算法公布不久，就有人找到了有效的分析方法，特别值得注意的是这些有效分析甚至包含了算法设计时试图避免的时间/存储/数据折中攻击[290-293]。

Sprout 算法的设计思想激发了其他新的轻量级流密码算法，例如 Fruit(Fruit 算法版本几经变动，较新的一个版本可参考文献[294])。Fruit 算法采用了 43 比特的 LFSR、37 比特的 NFSR、80 比特的固定密钥寄存器以及两个长度分别为 7 比特和 8 比特的计数器。但是该算法仍然不安全，存在 Lallemand-like 攻击[295]以及 ESC 2017 上公布的相关攻击，快速 Walsh 变换在攻击中起到了重要作用。Fruit 的设计者随后又对算法进行了调整，但仍然不能有效避免攻击。

Plantlet 是对于 Sprout 算法的另一个重要的改进，它继承了 Sprout 算法的基本结构，也是 Grain 系列算法族中的一员。Plantlet 设计的主要目的是修补 Sprout 算法的安全缺陷，以达到以下的设计目的：80 比特寄存器状态同时满足小面积要求和保证安全性，即使密钥永久存储并连续读取于可重写非易失存储器(Non-Volatile Memory，NVM)中，算法仍具有硬件友好性，不依赖于背后的 NVM 技术，能保持较高的吞吐率。Plantlet 算法采用了 61 比特 LFSR、40 比特 NFSR 和一个计数器，但对密钥的使用方式进行了简化设计。Plantlet 算法自公布以来，还没有人公布有效的分析方法。2017 年，Matthias Hamann 等学者研究了对于小状态流密码的时间/存储/数据折中分析，给出了 Plantlet 算法的一个分析

结果[296]。

尽管算法运行过程中使用依赖于密钥的状态更新函数在理论上具有一定的意义，而且可以减小硬件实现面积，但是在实际上不是这样。原因有两个。一是由于密钥并不是固化到器件上的，而是存储到 EEPROM 中的，所以持续访问密钥会减低 KSG 的时钟频率。特别当访问不是按顺序进行的时候，该问题尤其突出（例如在 Fruit 算法中）。同时，这种密钥访问的能量消耗也较大，RFID 标签执行一次 EEPROM 读操作，需要 5～10μW，执行一次写操作更是需要 50μW，而 RFID 标签的能量消耗预算仅为 10μW。为了避免使用 EEPROM 的缺点，可将密钥移到可变存储器中，但这又会使硬件实现面积增加。二是若密钥固化到器件上，虽然可以解决持续访问密钥的代价问题，但还需要考虑密钥的生命周期，即每个密钥可与多少初始向量搭配使用。

鉴于持续访问密钥带来的问题，Matthias Hamann 等人提出了 Lizard 算法。Lizard 算法也是 Grain 系列算法族中的一员，但是与 Sprout、Plantlet 等的设计思想不同，Lizard 算法采用 FP(1)模式，以获得对于时间/存储/数据折中攻击的可证明安全性，达到减少内部状态规模的目的（从 160 比特减少到 121 比特），而且由于同时采用了更强的输出函数与更大规模的密钥（从 80 比特增加到 120 比特），Lizard 算法目前并没有发现有效的分析方法。设计者认为 Lizard 算法可以避免频繁访问存储密钥的 EEPROM 带来的开销。Lizard 算法的设计安全性为 80 比特的状态恢复攻击安全性与 60 比特的区分攻击安全性，并且由于使用了 FP(1)模式，其对基于时间/存储/数据折中攻击的密钥恢复攻击的安全性可达到超越生日界的安全性 $2n/3$。

观察近年来轻量级流密码发展，可以看出其未来研究的一些端倪，主要有以下 4 点。一是对于资源受限的定义[297,298]。二是针对特殊应用，以优化的设计来达到安全轻量的目的。例如，Lizard 算法是针对小包模式的优化设计，即每个 Key/IV 对至多生成 2^{18} 比特密钥流（设计者称这足以满足 SSL/TLS、IEEE 802.11 等多数协议的要求）。三是可证明安全性，即一个轻量级流密码算法设计上的安全性是否是可证明的。四是关于密钥恢复或区分攻击，经典的时间/存储/数据折中攻击安全界是否可以改进。无论如何，相关问题研究与轻量级流密码的发展必会相互影响、相互促进[299]。

3.2 基于 LFSR 的流密码

3.2.1 A5/1

A5/1 是 GSM 网络中保障通信私密性的流密码算法。它最初是保密的，直到 1994 年才被逆向工程。A5 系列算法有多个变种。其中 A5/1 安全性最强，被用于欧洲和美国；A5/2 用于欧美以外地区；A5/3 是为第三代合作伙伴计划（3rd Generation Partnership Project，3GPP）设计的加密算法。由于这 3 个算法在实际应用中通常使用相同的会话密钥，攻破安全性较弱的算法会导致其余两个算法的安全问题。

A5/1 的内部状态为 64 比特，密钥长度为 64 比特，已知的初始化向量长度为 22 比特。完成内部状态初始化后，内部状态转移依赖一个择多（majority）钟控函数。A5/1 产生的密钥流为每帧通信的双方各提供 114 比特。

1. A5/1 的结构

A5/1 算法由 3 个长度分别为 19、22、23 比特的线性反馈移位寄存器 R1、R2、R3 组成，其结构如图 3-1 所示。其中，R1[8]、R2[10]和 R3[10]是钟控位，R1[13,16,17,18]、R2[20,21]和 R3[7,20,21,22]是反馈位。

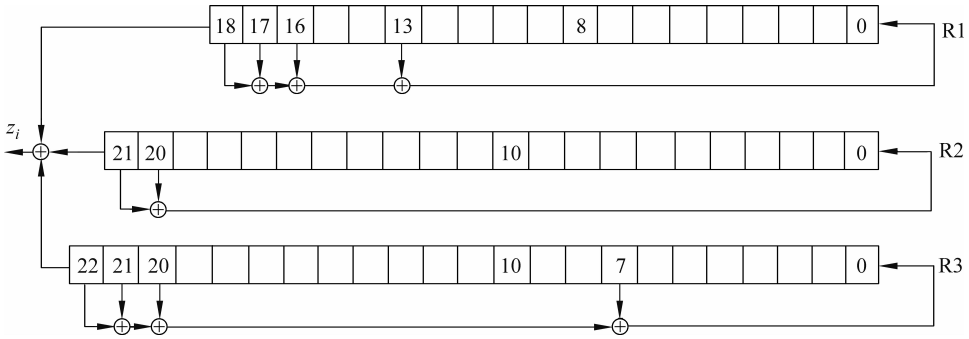

图 3-1　A5/1 的结构

2. A5/1 的伪随机比特生成算法

A5/1 的加密算法由内部状态初始化和伪随机比特生成两个阶段组成。内部状态初始化可用伪代码描述如下：

```
for i from 0 to 63 do
  R1,R2 and R3 are clocked
  R1[0] =R1[0] ⊕ K[i]
  R2[0] =R2[0] ⊕ K[i]
  R3[0] =R3[0] ⊕ K[i]
end for
for i from0 to 21 do
  R1,R2 and R3 are clocked
  R1[0] =R1[0] ⊕ Fn[i]
  R2[0] =R2[0] ⊕ Fn[i]
  R3[0] =R3[0] ⊕ Fn[i]
end for
```

初始化过程完成后的寄存器状态称为初始状态。

接下来要进行的是依赖择多钟控函数的停走。择多钟控函数为

$$\mathrm{Maj} = a \cdot b \oplus b \cdot c \oplus c \cdot a$$

该函数的输入为 R1[8]、R2[10]和 R3[10]，并且钟控位与 Maj 值一致的寄存器进行移位。进行 100 轮择多钟控停走后，继续进行 228 轮择多钟控停走，每轮停走后产生一个伪随机比特作为加解密的密钥流。每轮停走产生的密钥比特为

$$z_i = \mathrm{R1}[18] \oplus \mathrm{R2}[21] \oplus \mathrm{R3}[22]$$

上述过程用伪代码描述如下：

```
for i from 0 to 99 do
  R1,R2 and R3 are clocked with the majority-based clock control
end for
for i from 0 to 227 do
  R1,R2 and R3 are clocked with the majority-based clock control
  z_i =R1[18] ⊕ R2[21] ⊕ R3[22]
end for
```

3. A5/1 的加解密

A5/1 的加密算法是将明文比特串与 A5/1 生成的伪随机比特进行模 2 加法。解密算法是加密算法的逆。加解密过程如下：

$C_i = P_i \oplus z_i$

$P_i = C_i \oplus z_i$

4. A5/1 的安全性

A5/1 自 1994 年被逆向工程出来后得到了广泛的关注，经受了各种攻击的考验。1997 年，Golic 对 A5/1 进行了已知输出密钥流的分治攻击[300]。首先重建初始状态，然后由此确定密钥。分治攻击的平均计算复杂度是 $2^{40.16}$；然后又提出了基于生日悖论的时间/存储折中攻击。此攻击的目标是：对于已知输出密钥流，找出在某个已知时刻的未知内部状态。

2000 年，Biham 等人提出一种猜测确定攻击[301]。假设 R3 连续 10 轮没有进行状态更新，这样就可以得到大约 31 比特的寄存器信息。假设已知 R3[10]和 R3[22]，即可知这 10 轮中 R1 和 R2 的钟控位。猜测 R2[0]和 R1[9,10,11,12,14,15,16,17,18]来确定 R3 的未知比特。采用这种攻击方式，预计算复杂度是 $2^{33.6}$，数据复杂度是 $2^{20.8}$ 比特，存储空间是 2GB，时间复杂度是 $2^{40.97}$。

Biryukov 等人使用一台 PC 对 A5/1 进行了实时攻击[302]。他们用了两个方法：一是有偏差的生日攻击，二是随机子图攻击。攻击以 Golic 的时间/存储折中攻击为基础，利用 A5/1 内部状态的收缩，存储数量较少的初始状态，通过这些初始状态得到密钥。存储时利用易于采样的特殊状态来减少存储量。有偏差的生日攻击需要 2min 的通信数据、2(或 4)个 73GB 的硬盘和 2^{48}(或 2^{42})步预计算，可在 1s 内恢复出密钥。随机子图攻击只需要 2s 的通信数据，需要 4 个 73GB 硬盘和 2^{48} 步预计算，但是恢复密钥需要几分钟。

Barkan 等人首先介绍了对 A5/2 的唯密文攻击，利用纠错码和校验矩阵恢复密钥[303]。对 A5/1 的被动唯密文攻击也利用了相似的方法，同时结合了 Biryukov 等人的时间/存储折中攻击。Ekdahl 提出了对 A5/1 的一种相关攻击[304]。2008 年，Maximov 等人提出了改进的相关攻击[305]。与此同时，Gendrullis 等人利用特殊的硬件 COPACOBANA 实现了实际可用的攻击[306]。

Shah 等人进行了猜测确定攻击[307]。猜测 R1 的全部比特和输出密钥流比特，根据输出密钥流比特方程和钟控位之间的关系确定 R2 和 R3，成功率可达 100%。但是，在寻找过程中需要复制存储所有可能状态，需要的存储空间达 5.6GB。这种攻击的平均时间复杂度为 $2^{48.5}$。

Lu 等人提出了使用 GPGPU 的时间存储折中攻击，他们称这个分析为统一的彩虹表分

析。他们可以用 55 天完成一个 984GB 的彩虹表，如果使用两个这样的彩虹表，并且使用 8 个已知的 114 比特密钥流，就能够以 81%的成功率完成 9s 的在线攻击[308]。

3.2.2　E0

E0 是用于蓝牙的流密码算法，密钥长度可变，最长为 128 比特，初始向量(IV)长度为 74 比特(由 48 比特蓝牙地址和 26 比特主计数器构成)。蓝牙协议每帧最多处理 2745 比特，每一帧均由不同的初始向量加密，但是密钥在整个会话中保持不变。E0 和一般的流密码相同，均由内部状态初始化和密钥流比特生成两部分组成。为了增加生成的伪随机序列的长度和随机性，E0 采用了 4 个线性反馈移位寄存器。

1. E0 的结构

E0 由求和生成器演化而来，由线性反馈移位寄存器(4 个)、组合逻辑电路和混合器 3 部分构成，其结构如图 3-2 所示。线性反馈移位寄存器作为输入，并结合 4 比特内存单元得到密钥流。4 个线性反馈移位寄存器的长度分别为 25、31、33 和 39。它们的反馈多项式均为有 5 个非零项的本原多项式，分别为

$$P_1(x)=x^{25}+x^{20}+x^{12}+x^8+1$$
$$P_2(x)=x^{31}+x^{24}+x^{16}+x^{12}+1$$
$$P_3(x)=x^{33}+x^{28}+x^{24}+x^4+1$$
$$P_4(x)=x^{39}+x^{36}+x^{28}+x^4+1$$

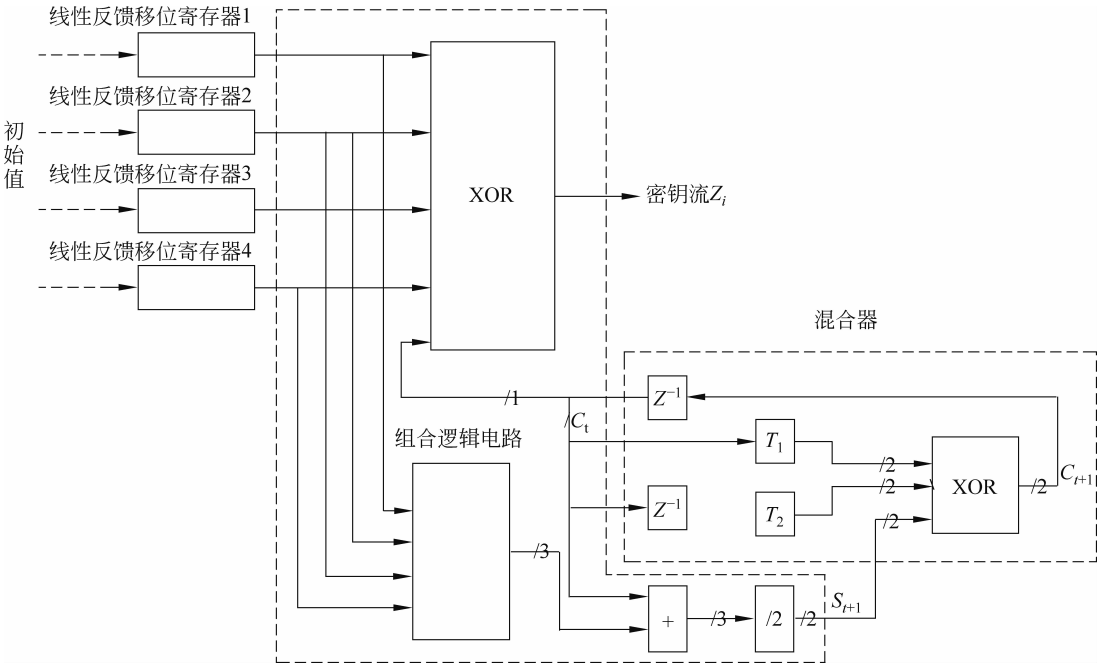

图 3-2　E0 的结构

混合器中 T_1 和 T_2 是线性变换网络，Z^{-1} 为延迟网络。C_t/S_t 时刻 t 的额外内存单元值。

2. E0 的内部状态初始化和密钥流生成

产生密钥流时，线性反馈移位寄存器需要赋初值。初始值进入线性反馈移位寄存器的过程如图 3-3 所示，此时 4 比特的额外内存单元置零。该初始化过程进行 200 步。在这 200 步中，后 128 比特再次作为线性反馈移位寄存器的初始值，额外内存单元的值 C_t 和 C_{t+1} 被保留，此时产生的输出即为通信密钥流。

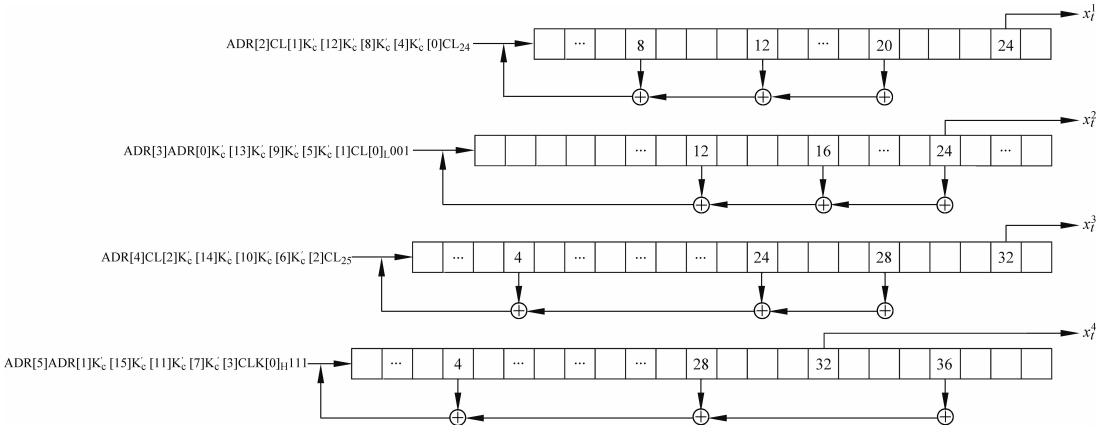

图 3-3　初始值进入 LFSR 的过程

其中，K_c' 由密钥 K_c 变换得到的，K_c 最长为 128 比特，ADR 为 48 比特主设备地址，CL 为 26 比特时钟，x_t^i 为时刻 t 第 i 个线性反馈移位寄存器的输出。

3. E0 的安全性

对 E0 的攻击最初出现在 2000 年前后，Hermelin 和 Nyberg 对蓝牙组合器进行了相关攻击，证明了给出 $O(2^{64})$ 长度的密钥流，128 比特密钥的蓝牙流密码可在 $O(2^{64})$ 步中被破解[309]。Fluhrer 等分析了 E0 加密模式[310]。

2004 年，Lu 等人对 E0 进行了区分攻击和密钥恢复攻击[311]。根据最大相关性的线性依赖进行了区分攻击，提出了基于快速 Walsh 变换的最大似然解码算法。同年，他们又发表了对两级 E0 的密钥恢复攻击，给出了 2^{35} 帧的前 24 比特，进行了 2^{40} 在线攻击[312]。文献[313]将密钥恢复攻击提升至进行复杂度为 2^{38} 的预计算，只需给出 $2^{23.8}$ 帧的前 24 比特。

2013 年，张斌等人提出了一种新的攻击，给出 $2^{22.7}$ 帧的前 24 比特，进行复杂度为 $2^{21.1}$ 的预计算，仅需 4MB 内存即可在复杂度为 2^{27} 的在线计算后恢复密钥[314]。2018 年，张斌等人又提出了一种降低数据复杂度的新方法。在已知初始向量的情景下，给出 2^{24} 帧的前 24 比特，进行复杂度为 $2^{21.1}$ 的预计算，仅需 4MB 内存即可在复杂度为 2^{25} 的在线计算后恢复密钥。他们将攻击变为唯密文攻击，给出 2^{26} 帧的前 24 比特，恢复密钥的总复杂度低于 2^{26}，这是目前对蓝牙加密模式的第一个实用的唯密文攻击[315]。

3.2.3　Snow 2.0

Snow 2.0 是 NESSIE 计划的候选算法之一。该算法面向软件设计，采用了线性反馈移位寄存器，结构比较简单，加解密速度快，固定初始向量为 128 比特，密钥长度有两种，分别是 128 比特和 256 比特。

1. Snow 2.0 算法

1) 加密流程

加密操作如下：首先是密钥初始化，它提供了线性反馈移位寄存器初始状态和有限状态机(Finite State Machine，FSM)内部寄存器 R1 和 R2 的初始值。然后是整个加密流程进行一次循环，生成一个密钥流字；接着再循环一次，生成一个密钥流字；后面依此类推，最终生成的密钥流字不超过 2^{50} 个。加密流程如图 3-4 所示。

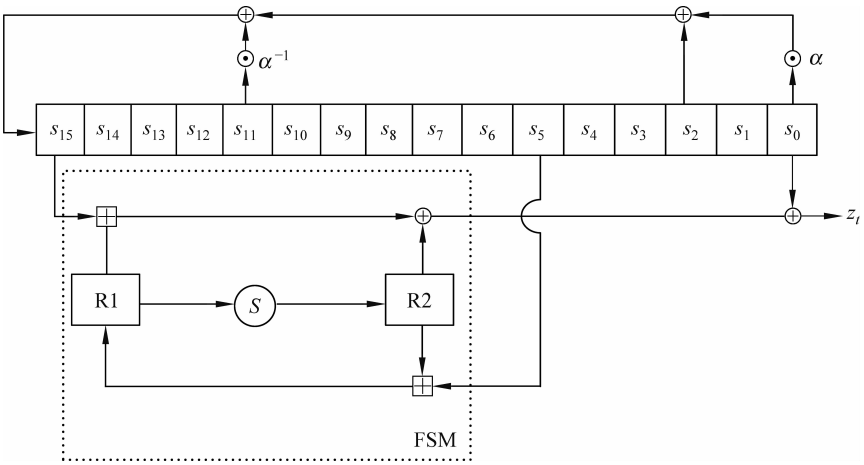

图 3-4　Snow 2.0 算法的加密流程

Snow 2.0 是面向 32 比特字的流密码算法，伪随机生成器由有限域 $F_{2^{32}}$ 上的 16 级线性反馈移位寄存器构成。反馈多项式为

$$\pi(x)=\alpha x^{16}+x^{14}+\alpha^{-1}x^{5}+1\in F_{2^{32}}[x]$$

其中，α 是以下多项式的根：

$$x^{4}+\beta^{23}x^{3}+\beta^{245}x^{2}+\beta^{48}x+\beta^{239}\in F_{2^{8}}[x]$$

β 是以下多项式的根：

$$x^{8}+x^{7}+x^{5}+x^{3}+1\in F_{2}[x]$$

非线性变换由 FSM 来实现，密钥初始化后，开始整个循环迭代。FSM 有两个寄存器，分别表示为 R1 和 R2，每个寄存器有 32 比特。在时刻 $t\geqslant 0$ 时，这两个寄存器的值分别表示为 $\mathrm{R1}_t$ 和 $\mathrm{R2}_t$。FSM 的输入是(s_{t+15},s_{t+5})，输出用 F_t 表示，F_t 的计算公式如下：

$$F_t=(s_{t+15}\boxplus \mathrm{R1}_t)\oplus \mathrm{R2}_t,\quad t\geqslant 0$$

输出的密钥流用 z_t 表示，z_t 的计算公式如下：

$$z_t=F_t\oplus s_t,\quad t\geqslant 1$$

而寄存器 R1 和 R2 更新如下：

$$\mathrm{R1}_{t+1}=s_{t+5}\boxplus R2_t$$
$$\mathrm{R2}_{t+1}=S(\mathrm{R1}_t),\quad t\geqslant 0$$

2) S 盒

S 盒记作 $S(w)$，是一个输入和输出均为 32 比特的非线性变换，它由 4 个并置的 8 比特 S 盒加换位变换构成。用 w 表示 S 盒的输入：

$$w=\begin{bmatrix}w_0\\w_1\\w_2\\w_3\end{bmatrix}$$

应用 AES 的 S 盒，w 变成

$$w=\begin{bmatrix}S_R[w_0]\\S_R[w_1]\\S_R[w_2]\\S_R[w_3]\end{bmatrix}$$

然后，进行列混合变换并输出 r：

$$\begin{bmatrix}r_0\\r_1\\r_2\\r_3\end{bmatrix}=\begin{bmatrix}x & x+1 & 1 & 1\\1 & x & x+1 & 1\\1 & 1 & x & x+1\\x+1 & 1 & 1 & x\end{bmatrix}\begin{bmatrix}S_R[w_0]\\S_R[w_1]\\S_R[w_2]\\S_R[w_3]\end{bmatrix}$$

即 $r=S(w)$。

3）密钥初始化

初始向量记作 $\mathrm{IV}=(\mathrm{IV}_3,\mathrm{IV}_2,\mathrm{IV}_1,\mathrm{IV}_0)$，128 比特密钥记作 $K=(k_3,k_2,k_1,k_0)$，256 比特密钥记作 $K=(k_7,k_6,k_5,k_4,k_3,k_2,k_1,k_0)$。

128 比特密钥初始化如下：

$s_{15}=k_3\oplus\mathrm{IV}_0$，$s_{14}=k_2$，$s_{13}=k_1$，$s_{12}=k_0\oplus\mathrm{IV}_1$，
$s_{11}=k_3\oplus 1$，$s_{10}=k_2\oplus 1\oplus\mathrm{IV}_2$，$s_9=k_1\oplus 1\oplus\mathrm{IV}_3$ $s_8=k_0\oplus 1$，
$s_7=k_3$，$s_6=k_2$，$s_5=k_1$，$s_4=k_0$，
$s_3=k_3\oplus 1$，$s_2=k_2\oplus 1$，$s_1=k_1\oplus 1$，$s_0=k_0\oplus 1$

其中 1 表示 32 比特全 1 向量。

256 比特密钥初始化与上面类似：

$s_{15}=k_7\oplus\mathrm{IV}_0$ $s_{14}=k_6$ $s_{13}=k_5$ $s_{12}=k_4\oplus\mathrm{IV}_1$
$s_{11}=k_3$ $s_{10}=k_2\oplus\mathrm{IV}_2$ $s_9=k_1\oplus\mathrm{IV}_3$ $s_8=k_0$
$s_7=k_7\oplus 1$ $s_6=k_6\oplus 1$ … $s_0=k_0\oplus 1$

在初始化过程中循环 32 轮，但不生成输出密钥流。Snow 2.0 的初始化流程如图 3-5 所示。

2. Snow 2.0 的安全性

Snow 2.0 虽然针对 Snow 1.0 的弱点做了很大的改进，但是仍存在代数攻击、线性区分攻击、快速相关攻击等。其中区分攻击针对 Snow 2.0 所需的时间复杂度为 2^{174}，数据复杂度为 2^{174}[316]。最新的快速相关攻击比原来的密钥恢复攻击已经有了很大的突破[317]，所需的时间复杂度为 $2^{164.15}$，数据复杂度为 $2^{163.59}$。Snow 2.0 算法是欧洲 eSTREAM 计划的指定比较算法，国际密码学界普遍认为 Snow 2.0 是一个 128 比特密钥下的十分安全且高效的流密码算法。

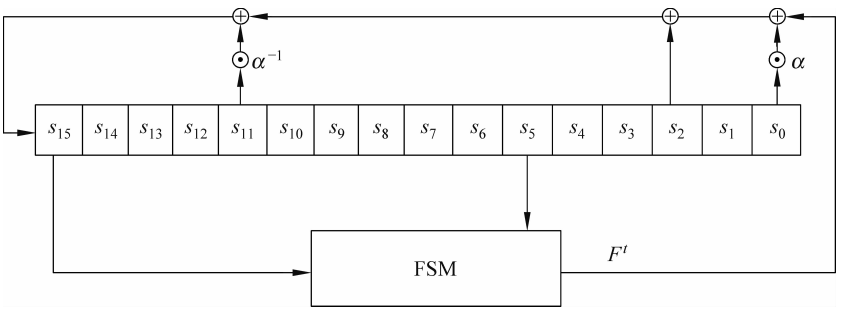

图 3-5　Snow 2.0 的初始化流程

3.2.4　Snow 3G

Snow 3G 是 Snow 1.0 和 Snow 2.0 的设计者为抵抗代数攻击而推出的又一个流密码算法。由于其结构简洁，加解密速度快，安全性高，最终被选入 3GPP 的机密性和完整性机制 UEA2 和 UIA2，成为第四代移动通信加密标准。Snow 3G 是一个面向字的流密码算法，在 128 比特密钥和 128 比特初始向量的控制下，生成以 32 比特字为单位的密钥流序列。Snow 3G 的执行分为两个阶段：初始化阶段和工作阶段。第一阶段执行密钥初始化，即在加密循环过程中不输出密钥流；第二阶段每轮循环都会输出一个 32 比特密钥流。

1. Snow 3G 算法组成部分

1) MULx

MULx 是从 16 比特到 8 比特的映射。令 V 和 c 为 8 比特的值，则 MULx 定义如下：

如果 V 的最左边(最高有效)比特为 1，则

$$\mathrm{MULx}(V,c)=(V\ll_8 1)\oplus c$$

否则，

$$\mathrm{MULx}(V,c)=V\ll_8 1$$

2) MULxPOW

MULxPOW 是从 16 比特和一个正整数 i 到 8 比特的映射。令 V 和 c 为 8 比特的值，则 MULxPOW(V,i,c)递归定义如下：

如果 i 为 0，则

$$\mathrm{MULxPOW}(V,i,c)=V$$

否则，

$$\mathrm{MULxPOW}(V,i,c)=\mathrm{MULx}(\mathrm{MULxPOW}(V,i-1,c),c)$$

3) 32 比特 S 盒 S_1

S_1 是 32 比特输入、32 比特输出的非线性映射。令 $w=w_0\parallel w_1\parallel w_2\parallel w_3$ 为 32 比特输入，w_0 和 w_3 分别为最高有效字节和最低有效字节。使用 AES 的 8 比特 S 盒 S_R，如下计算 32 比特输出 $S_1(w)=r_0\parallel r_1\parallel r_2\parallel r_3$。

$$r_0=\mathrm{MULx}(S_R(w_0),0\mathrm{x}1\mathrm{B})\oplus S_R(w_1)\oplus S_R(w_2)\oplus \mathrm{MULx}(S_R(w_3),0\mathrm{x}1\mathrm{B})\oplus S_R(w_3)$$

$$r_1=\mathrm{MULx}(S_R(w_0),0\mathrm{x}1\mathrm{B})\oplus S_R(w_0)\oplus \mathrm{MULx}(S_R(w_1),0\mathrm{x}1\mathrm{B})\oplus S_R(w_2)\oplus S_R(w_3)$$

$$r_2=S_R(w_0)\oplus \mathrm{MULx}(S_R(w_1),0\mathrm{x}1\mathrm{B})\oplus S_R(w_1)\oplus \mathrm{MULx}(S_R(w_2),0\mathrm{x}1\mathrm{B})\oplus S_R(w_3)$$

$r_3 = S_R(w_0) \oplus S_R(w_1) \oplus \mathrm{MULx}(S_R(w2),0\mathrm{x}1\mathrm{B}) \oplus S_R(w_2) \oplus \mathrm{MULx}(S_R(w_3),0\mathrm{x}1\mathrm{B})$

4) 32 比特 S 盒 S_2

S_2 是 32 比特输入、32 比特输出的非线性映射。令 $w = w_0 \| w_1 \| w_2 \| w_3$ 为 32 比特输入，使用基于 Dickson 多项式导出的 8 比特 S 盒 S_Q，如下计算 $S_2(w) = r_0 \| r_1 \| r_2 \| r_3$ 32 比特输出：

$r_0 = \mathrm{MULx}(S_Q(w_0),0\mathrm{x}69) \oplus S_Q(w_1) \oplus S_Q(w_2) \oplus \mathrm{MULx}(S_Q(w3),0\mathrm{x}69) \oplus S_Q(w_3)$

$r_1 = \mathrm{MULx}(S_Q(w_0),0\mathrm{x}69) \oplus S_Q(w_0) \oplus \mathrm{MULx}(S_Q(w1),0\mathrm{x}69) \oplus S_Q(w_2) \oplus S_Q(w_3)$

$r_2 = S_Q(w_0) \oplus \mathrm{MULx}(S_Q(w1),0\mathrm{x}69) \oplus S_Q(w_1) \oplus \mathrm{MULx}(S_Q(w2),0\mathrm{x}69) \oplus S_Q(w_3)$

$r_3 = S_Q(w_0) \oplus S_Q(w_1) \oplus \mathrm{MULx}(S_Q(w2),0\mathrm{x}69) \oplus S_Q(w_2) \oplus \mathrm{MULx}(S_Q(w3),0\mathrm{x}69)$

5) 函数 MUL_α

函数 MUL_α 是 8 比特到 32 比特的映射。令 c 为 8 比特输入，则 MUL_α 定义如下：

$$\mathrm{MULa}(c) = (\mathrm{MULxPOW}(c,23,0\mathrm{xA9}) \| \mathrm{MULxPOW}(c,245,0\mathrm{xA9}) \| \mathrm{MULxPOW}(c,48,0\mathrm{xA9}) \| \mathrm{MULxPOW}(c,239,0\mathrm{xA9}))$$

6) 函数 DIV_α

函数 DIV_α 是 8 比特到 32 比特的映射。令 c 为 8 比特输入，则 DIV_α 定义如下：

$$\mathrm{DIV}_\alpha(c) = (\mathrm{MULxPOW}(c,16,0\mathrm{xA9}) \| \mathrm{MULxPOW}(c,39,0\mathrm{xA9}) \| \mathrm{MULxPOW}(c,6,0\mathrm{xA9}) \| \mathrm{MULxPOW}(c,64,0\mathrm{xA9}))$$

7) 初始化模式

LFSR 的初始化模式接收来自 FSM 的 32 比特输出字 F。

令 $s_0 = s_{0,0} \| s_{0,1} \| s_{0,2} \| s_{0,3}$，$s_{11} = s_{11,0} \| s_{11,1} \| s_{11,2} \| s_{11,3}$，其中 $s_{11,0}$ 和 $s_{11,3}$ 分别为最高有效字节和最低有效字节。如下计算中间值 v：

$$v = (s_{0,1} \| s_{0,2} \| s_{0,3} \| 0\mathrm{x}00) \oplus \mathrm{MUL}_\alpha(s_{0,0}) \oplus s_2 \oplus (0\mathrm{x}00 \| s_{11,0} \| s_{11,1} \| s_{11,2}) \oplus \mathrm{DIV}_\alpha(s_{11,3}) \oplus F$$

令

$$s_0 = s_1,\ s_1 = s_2,\ s_2 = s_3,\ s_3 = s_4,\ s_4 = s_5,\ s_5 = s_6,\ s_6 = s_7,\ s_7 = s_8,$$
$$s_8 = s_9,\ s_9 = s_{10},\ s_{10} = s_{11},\ s_{11} = s_{12},\ s_{12} = s_{13},\ s_{13} = s_{14},\ s_{14} = s_{15},\ s_{15} = v$$

8) FSM 循环模式

FSM 有两个来自 LFSR 的输入字 s_{15} 和 s_5，输出一个 32 比特的字 F：

$$F = (s_{15} \boxplus \mathrm{R1}) \oplus \mathrm{R2}$$

其中⊞是模 2^{32} 加法。

然后寄存器更新，计算中间值 r：

$$r = \mathrm{R2} \boxplus (\mathrm{R3} \oplus s_5)$$

令

$$\mathrm{R3} = S_2(\mathrm{R2})$$
$$\mathrm{R2} = S_1(\mathrm{R1})$$
$$\mathrm{R1} = r$$

2. Snow 3G 算法操作

1) 初始化

Snow 3G 初始化使用 128 比特密钥和 128 比特初始向量，密钥包含 4 个 32 比特字 k_0，k_1，k_2，k_3，初始向量包含 4 个 32 比特字 IV_0，IV_1，IV_2，IV_3，定义如下：

$s_{15}=k_3\oplus \mathrm{IV0}$，　$s_{14}=k_2$，　$s_{13}=k_1$，　$s_{12}=k_0\oplus \mathrm{IV}_1$，

$s_{11}=k_3\oplus 1$，　$s_{10}=k_2\oplus 1\oplus \mathrm{IV}_2$，　$s_9=k_1\oplus 1\oplus \mathrm{IV}_3$，　$s_8=k_0\oplus 1$，

$s_7=k_3$，　$s_6=k_2$，　$s_5=k_1$，　$s_4=k_0$，

$s_3=k_3\oplus 1$，　$s_2=k_2\oplus 1$，　$s_1=k_1\oplus 1$，　$s_0=k_0\oplus 1$

其中 1 表示全 1 的字(0xFFFFFFFF)。FSM 初始化为 R1 = R2 = R3 = 0。

然后加密，以一种特殊模式运行，不产生输出密钥流，伪代码描述如下：

```
for t from 1 to 32 do
    1. The FSM is clocked producing the 32-bit word F.
    2. Then the LFSR is clocked in Initialisation Mode using F.
end for
```

Snow 3G 的初始化流程如图 3-6 所示。

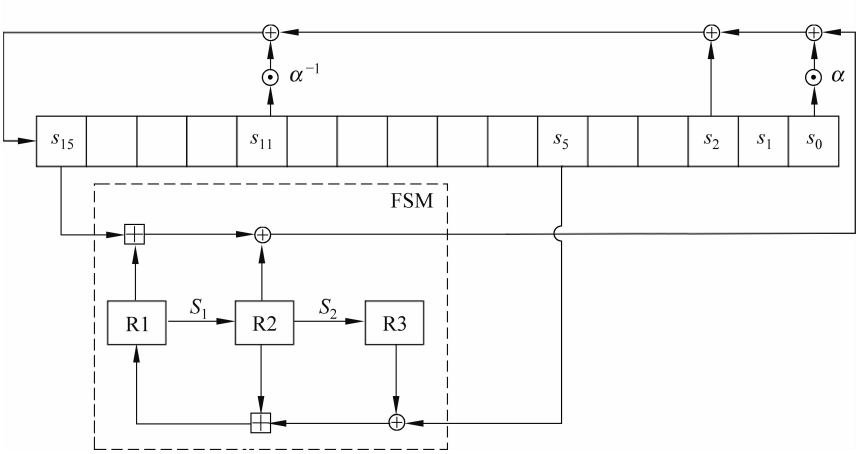

图 3-6　Snow 3G 的初始化流程

2）密钥流生成

首先 FSM 循环执行一次，但是丢弃 FSM 的输出字。然后 LFSR 在密钥流初始化模式执行一次。最后产生加密使用的 n 个 32 比特的密钥流字，伪代码描述如下：

```
for t from 1 to n do
    3. The FSM is clocked and produces a 32-bit output word F.
    4. The next keystream word is computed as z_t = F ⊕ s_0.
    5. Then the LFSR is clocked in Keystream Mode.
end for
```

Snow 3G 的密钥流生成的流程如图 3-7 所示。

3. Snow 3G 安全性分析

到目前为止，还没有一种行之有效的方法可以完全破译 Snow 3G，但针对它的改进版本的攻击已经可以在 $2^{124.2}$ 复杂度下达到 16 轮(初始化共 33 轮)[318]。针对 Snow 3G 算法的密钥流生成阶段，目前最好的状态恢复攻击是文献[319]给出的结果，复杂度为 2^{177}。

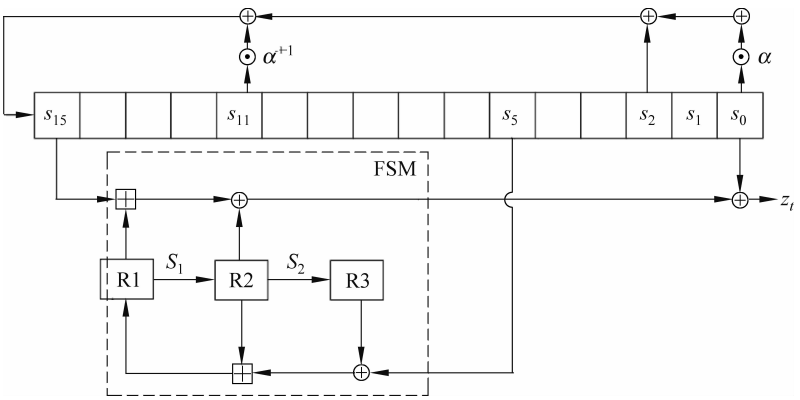

图 3-7　Snow 3G 的密钥流生成流程

3.2.5　ZUC

ZUC 算法(又称祖冲之算法)是我国自主设计的流密码算法,2011 年被 3GPP LTE 采纳为国际加密标准,即第四代移动通信加密标准。ZUC 算法同时也是中国第一个成为国际标准的密码算法,其标准化的成功是中国在商用密码算法领域取得的一次重大突破。与早期同样被纳入 3GPP LTE 国际加密标准的 Snow 3G 相比,在吞吐率相同的情形下,ZUC 的资源和功耗更小。

ZUC 是面向字的流密码算法。它采用 128 比特初始密钥和 128 比特初始向量作为输入,执行算法后,输出一个 32 比特字的密钥流。通信中使用此密钥流进行加密和解密。ZUC 的执行分为两个阶段:初始化阶段和工作阶段。第一阶段,执行密钥和初始向量的初始化,不产生密钥流输出;第二阶段,每一轮加密循环生成一个 32 比特密钥流。

1. ZUC 算法描述

1) 算法的一般结构

ZUC 有 3 个逻辑层,如图 3-8 所示。顶层是一个 16 级的线性反馈移位寄存器(LFSR),中间层是比特重组层(BR),底层是一个非线性函数 F(FSM)。

2) 线性反馈移位寄存器

线性反馈移位寄存器(LFSR)有 16 个 31 比特的单元($s_0, s_2, \cdots, s_{15}$)。每个单元 s_i($0 \leqslant i \leqslant 15$)限定只能从集合$\{1, 2, \cdots, 2^{31}-1\}$中取值。LFSR 有两个操作模式:初始化模式和工作模式。

在初始化模式下,LFSR 收到一个 31 比特字 u,它是由非线性函数 F 产生的 32 比特字 w 移除最右边 1 比特得到的,即 $u = w \gg 1$。伪代码描述如下:

```
LFSRWithInitializationMode(u)
{
    v = (2^15 s_15 + 2^17 s_13 + 2^21 s_10 + 2^20 s_4 + (1+2^8) s_0) mod (2^31 - 1)
    s_16 = (v + u) mod (2^31 - 1)
    if s_16 = 0   then set s_16 = 2^31 - 1
    (s_1, s_2, ···, s_16) → (s_0, s_1, ···, s_15)
}
```

在工作模式下，LFSR 不会收到任何输入，伪代码描述如下：

```
LFSRWithWorkMode(u)
{
    s16=(2^15 s15+2^17 s13+2^21 s10+2^20 s4+(1+2^8) s0) mod (2^31-1)
    if s16=0,   then set s16=2^31-1
    (s1,s2,…,s16) → (s0,s1,…,s15)
}
```

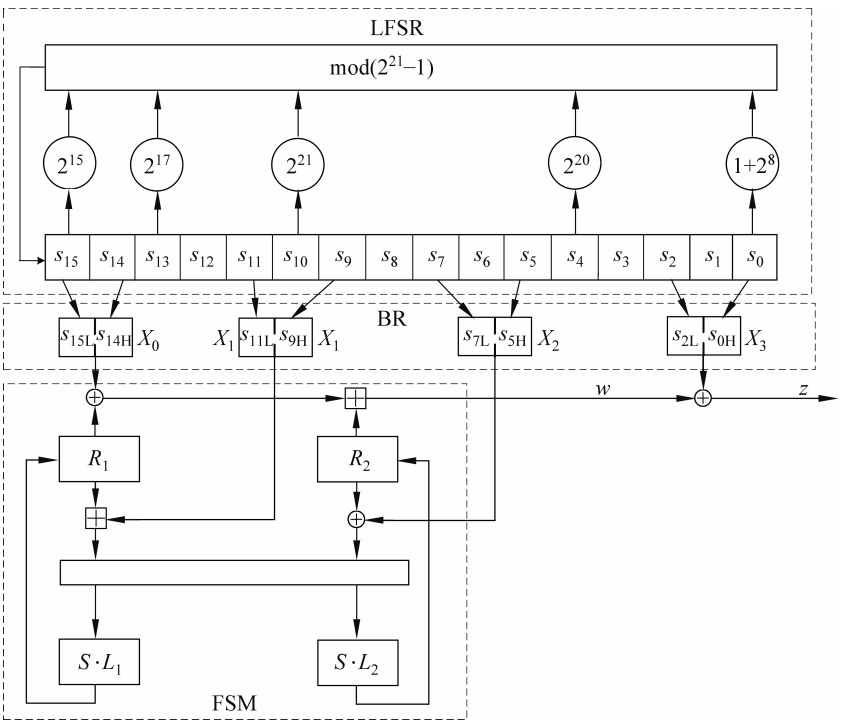

图 3-8　ZUC 算法的结构

提示 1：31 比特串 s 在 GF($2^{31}-1$)上乘以 2^i 可以通过循环左移 i 比特实现，比如，在 LFSR 初始化模式第一步中，可以转换成下面的式子：

$$v=(s_{15}\lll_{31}15)+(s_{13}\lll_{31}17)+(s_{10}\lll_{31}21)+(s_4\lll_{31}20)+(s_0\lll_{31}8)+s_0 \bmod(2^{31}-1)$$

提示 2：对于在 GF($2^{31}-1$)上的两个整数 a 和 b，$v=(a+b) \bmod (2^{31}-1)$可以由下面两步实现：①计算 $v=a+b$；②如果进位是 1，则设定 $v=v+1$。另外，为了更好地抵抗可能的计时攻击，有下面两个补充：①计算 $w=a+b$，其中 w 是一个 32 比特的值；②设定 $v=w$ 的最左边 31 位比特$+w$ 的最右边的比特。

3）比特重组

ZUC 算法的中间层是比特重组。它从 LFSR 中抽取 128 比特，生成 4 个 32 比特的字，前 3 个字用于底层非线性函数 F，最后 1 个字参与密钥流生成。

令 $s_0, s_2, s_5, s_7, s_9, s_{11}, s_{14}, s_{15}$ 为上述 LFSR 的 8 个单元。比特重组就是使用它们形成 4 个 32 比特的字 X_0, X_1, X_2, X_3，伪代码描述如下：

```
Bitreorganization()
{
    X0=S15H || S14L
    X1=S11L || S9H
    X2=S7L || S5H
    X3=S2L || S0H
}
```

提示：s_i 是 31 比特，所以 $s_{i\mathrm{H}}$ 意味着比特下标为 30～15，而不是 31～16，其中 $0 \leqslant i \leqslant 15$。

4）非线性函数 F

非线性函数 F 有两个 32 比特内存单元 R_1 和 R_2。令 F 的输入为 X_0, X_1, X_2，它们来自比特重组的输出，经过 F 函数后生成一个 32 比特字 W。伪代码描述如下：

```
F(X0,X1,X2)
{
    W = (X0 ⊕ R1) ⊞ R2
    W1=R1⊞X1
    W2=R2⊞X2
    R1=S(L1(W1L || W2H))
    R2=S(L2(W2L || W1H))
}
```

其中，S 是一个 32 比特 S 盒：L_1 和 L_2 都是从 32 比特到 32 比特的线性变换，具体如下：

$$L_1(X) = X \oplus (X \lll_{32} 2) \oplus (X \lll_{32} 10) \oplus (X \lll_{32} 18) \oplus (X \lll_{32} 24)$$

$$L_2(X) = X \oplus (X \lll_{32} 8) \oplus (X \lll_{32} 14) \oplus (X \lll_{32} 22) \oplus (X \lll_{32} 30)$$

32 比特 S 盒 S 由 4 个 8 比特 S 盒组成，即 $S = (S_0, S_1, S_2, S_3)$，其中 $S_0 = S_2, S_1 = S_3$。S_0 和 S_1 见表 3-1 和表 3-2。令 x 为 S_0（或者 S_1）的一个 8 比特输入，把 x 写成两个 16 进制的形式：$x = h \| l$，那么在下表中第 h 行第 l 列的单元即为 S_0（或者 S_1）的输出。

表 3-1　S_0

行＼列	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	3E	72	5B	47	CA	E0	00	33	04	D1	54	98	09	B9	6D	CB
1	7B	1B	F9	32	AF	9D	6A	A5	B8	2D	FC	1D	08	53	03	90

续表

行＼列	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	4D	4E	84	99	E4	CE	D9	91	DD	B6	85	48	8B	29	6E	AC
3	CD	C1	F8	1E	73	43	69	C6	B5	BD	FD	39	63	20	D4	38
4	76	7D	B2	A7	CF	ED	57	C5	F3	2C	BB	14	21	06	55	9B
5	E3	EF	5E	31	4F	7F	5A	A4	0D	82	51	49	5F	BA	58	1C
6	4A	16	D5	17	A8	92	24	1F	8C	FF	D8	AE	2E	01	D3	AD
7	3B	4B	DA	46	EB	C9	DE	9A	8F	87	D7	3A	80	6F	2F	C8
8	B1	B4	37	F7	0A	22	13	28	7C	CC	3C	89	C7	C3	96	56
9	07	BF	7E	F0	0B	2B	97	52	35	41	79	61	A6	4C	10	FE
A	BC	26	95	88	8A	B0	A3	FB	C0	18	94	F2	E1	E5	E9	5D
B	D0	DC	11	66	64	5C	EC	59	42	75	12	F5	74	9C	AA	23
C	0E	86	AB	BE	2A	02	E7	67	E6	44	A2	6C	C2	93	9F	F1
D	F6	FA	36	D2	50	68	9E	62	71	15	3D	D6	40	C4	E2	0F
E	8E	83	77	6B	25	05	3F	0C	30	EA	70	B7	A1	E8	A9	65
F	8D	27	1A	DB	81	B3	A0	F4	45	7A	19	DF	EE	78	34	60

表 3-2　S_1

行＼列	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	55	C2	63	71	3B	C8	47	86	9F	3C	DA	5B	29	AA	FD	77
1	8C	C5	94	0C	A6	1A	13	00	E3	A8	16	72	40	F9	F8	42
2	44	26	68	96	81	D9	45	3E	10	76	C6	A7	8B	39	43	E1
3	3A	B5	56	2A	C0	6D	B3	05	22	66	BF	DC	0B	FA	62	48
4	DD	20	11	06	36	C9	C1	CF	F6	27	52	BB	69	F5	D4	87
5	7F	84	4C	D2	9C	57	A4	BC	4F	9A	DF	FE	D6	8D	7A	EB
6	2B	53	D8	5C	A1	14	17	FB	23	D5	7D	30	67	73	08	09
7	EE	B7	70	3F	61	B2	19	8E	4E	E5	4B	93	8F	5D	DB	A9
8	AD	F1	AE	2E	CB	0D	FC	F4	2D	46	6E	1D	97	E8	D1	E9
9	4D	37	A5	75	5E	83	9E	AB	82	9D	B9	1C	E0	CD	49	89
A	01	B6	BD	58	24	A2	5F	38	78	99	15	90	50	B8	95	E4
B	D0	91	C7	CE	ED	0F	B4	6F	A0	CC	F0	02	4A	79	C3	DE

续表

行＼列	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C	A3	EF	EA	51	E6	6B	18	EC	1B	2C	80	F7	74	E7	FF	21
D	5A	6A	54	1E	41	31	92	35	C4	33	07	0A	BA	7E	0E	34
E	88	B1	98	7C	F3	3D	60	6C	7B	CA	D3	1F	32	65	04	28
F	64	BE	85	9B	2F	59	8A	D7	B0	25	AC	AF	12	03	E2	F2

2. 密钥加载

密钥加载程序扩展初始密钥和初始向量为 16 个 31 比特的整数，作为 LFSR 的初始状态。令 128 比特的初始密钥 k 和 128 比特的初始向量 iv 为

$$k = k_0 \parallel k_1 \parallel \cdots \parallel k_{15}$$

$$\mathrm{iv} = \mathrm{iv}_0 \parallel \mathrm{iv}_1 \parallel \cdots \parallel \mathrm{iv}_{15}$$

其中 k_i 和 iv_i 都是字节($0 \leqslant i \leqslant 15$)，那么 k 和 iv 按下面两个规则加载到 LFSR。

(1) 令 D 是一个 240 比特常量字符串，由 16 个 15 比特长的子串构成：

$$D = d_0 \parallel d_1 \parallel \cdots \parallel d_{15}$$

其中

$$d_0 = 100010011010111_2$$

$$d_1 = 010011010111100_2$$

$$d_2 = 110001001101011_2$$

$$d_3 = 001001101011110_2$$

$$d_4 = 101011110001001_2$$

$$d_5 = 011010111100010_2$$

$$d_6 = 111000100110101_2$$

$$d_7 = 000100110101111_2$$

$$d_8 = 100110101111000_2$$

$$d_9 = 010111100010011_2$$

$$d_{10} = 110101111000100_2$$

$$d_{11} = 001101011110001_2$$

$$d_{12} = 101111000100110_2$$

$$d_{13} = 011110001001101_2$$

$$d_{14} = 111100010011010_2$$

$$d_{15} = 100011110101100_2$$

(2) 对于 $0 \leqslant i \leqslant 15$，令 $s_i = k_i \parallel d_i \parallel \mathrm{iv}_i$。

3. ZUC 算法执行

ZUC 算法的执行有两个阶段：初始化阶段和工作阶段

1) 初始化阶段

在初始化阶段，ZUC 算法调用密钥加载程序把 128 比特的初始密钥和 128 比特的初始

向量加载到 LFSR，再把 32 比特的内存单元 R_1 和 R_2 都设为 0。然后 ZUC 算法运行下列操作 32 次：

```
Bitreorganization()
W =F (X0,X1,X2)
LFSRWithInitializationMode(W≫1)
```

2）工作阶段

初始化阶段后，ZUC 算法转移到工作阶段。在工作阶段，ZUC 算法执行下列操作一次，并且把 F 的输出 w 丢弃：

```
Bitreorganization()
W =F (X0,X1,X2)
LFSRWithWorkMode()
```

然后 ZUC 算法进入密钥流生成阶段，即在每次迭代中执行一次下列操作，32 比特的密钥流 Z 就会生成并输出：

```
Bitreorganization()
Z =F (X0,X1,X2) ⊕ X3
LFSRWithWorkMode()
```

3.3　基于 NFSR 的流密码

3.3.1　A2U2

A2U2 是为印刷电子标签而设计的一种轻量级流密码算法。A2U2 由一个线性反馈移位寄存器(LFSR)、两个非线性反馈移位寄存器(NFSR)、一个密钥调度和一个过滤函数四部分构成，它的密钥长度是 56 比特。

1. A2U2 的结构

A2U2 是由一个 7 级线性反馈移位寄存器、两个分别长 17 比特和 9 比特的非线性反馈移位寄存器、一个密钥调度模块和一个过滤函数 4 部分构成，如图 3-9 所示。

A2U2 的基本模块定义如下。

1）线性反馈移位寄存器

7 级 LFSR 作为计数器，反馈函数为 $F_c = x^7 + x^4 + 1$。计数器由以下 3 个比特串的异或值初始化：由标签生成的 32 比特伪随机数的 5 个 LSB(最低有效位)、读取器生成的 32 比特随机数的 5 个 LSB 和密钥的 5 个 LSB。5 个比特的结果输入到 LFSR 的 MSB(最高有效位)。为避免全零状态，将 LFSR 的第二个 LSB 置 1；为避免全 1 状态，要将第一个 LSB 置 0。

初始化直到 LFSR 状态变为全 1 时为止。根据随机选择的计数器初始化值，计数器被计时的次数(9～126 次)是不规则和秘密的。在初始化完成后，计数器将只作为 LFSR 使用。

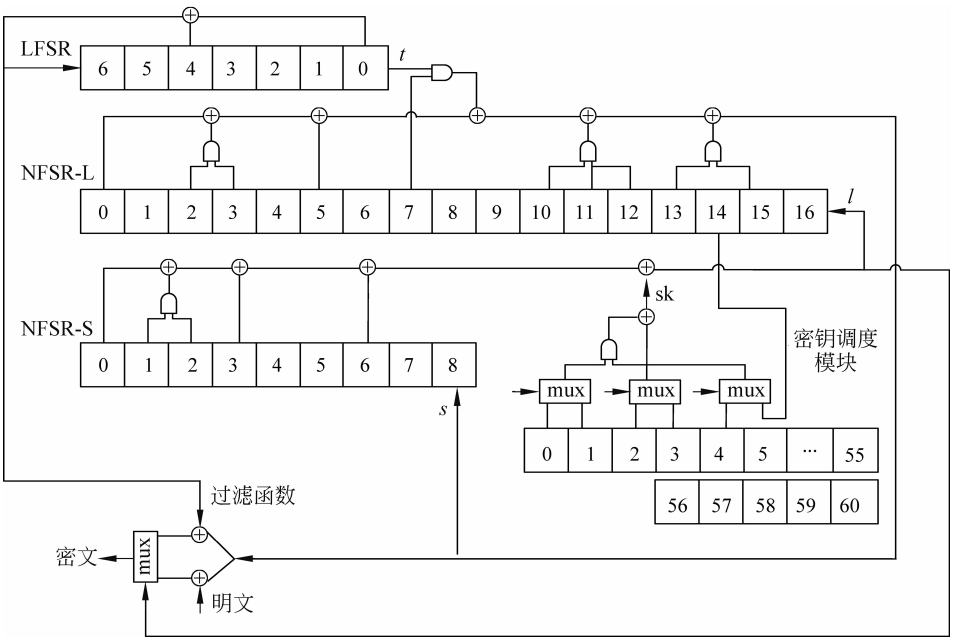

图 3-9　A2U2 的结构

2）非线性反馈移位寄存器

两个 NFSR 互相为对方提供反馈。在时刻 i，两个 NFSR 的反馈函数分别为

$$s_{i+9}=l_i+l_{i+2}\cdot l_{i+3}+l_{i+5}+l_{i+7}\cdot t_i+l_{i+10}\cdot l_{i+11}\cdot l_{i+12}+l_{i+13}\cdot l_{i+15}$$

$$l_{i+17}=s_i+s_{i+1}\cdot s_{i+2}+s_{i+3}+s_{i+6}+\mathrm{sk}_i$$

其中，sk_i 是密钥调度模块生成的子密钥比特。

3）密钥调度模块

A2U2 的密钥部分有 61 比特，前 56 比特用于产生子密钥，后 5 比特为计数器初始化保留。在时刻 i，56 比特密钥中的 5 个比特、计数器的 3 个比特和 NFSR-L 的 1 个比特组合产生子密钥 sk_i：

$$\mathrm{sk}_i=\mathrm{mux}(k_{5i \bmod 56},k_{(5i+1) \bmod 56},t_{i+1})\times \mathrm{mux}(k_{(5i+4) \bmod 56},l_{i+14},t_{i+5})+\mathrm{mux}(k_{(5i+2) \bmod 56},k_{(5i+3) \bmod 56},t_{i+3})$$

其中，mux()是复用器。对任意给出的 $x,y,z\in F_2$，如果 $z=0$，那么 $\mathrm{mux}(x,y,z)=x$；否则，$\mathrm{mux}(x,y,z)=y$。

4）过滤函数 F

假设在时刻 0 产生密文，过滤器定义如下：

$$c_i=\mathrm{mux}(s_{i+8}+t_{i+6},s_{i+8}+p_{f(i)},l_{i+16})$$

p_i 和 c_i 是时刻 i 的明密文比特，$f(0)=0$，$i\geqslant 1$ 时，$f(i)=\sum_{j=0}^{i-1}l_{j+16}$。

2. A2U2 的安全性

2011 年，Chai 等人提出了一种针对 A2U2 的选择性明文攻击模型下的高效密钥恢复攻击，该攻击要求两个选择的明文使用相同的密钥和初始向量进行加密[320]。Abdelraheem